Instavine

Manan Vyas

Nicholas Yingst

Shruti Sharma

1. ABSTRACT:

The Instagram app went viral soon after its launch. It became so popular that Facebook decided to take over it. The motivation for this project is to take the features of Instagram one step ahead and create a real-time video Instagram. The user will be able to choose from a given set of filters that can be applied to the video that is being captured by the camera in real-time. So it is a fusion of the popular Image App 'Instagram®' and the video app 'Vine®' and hence, we decided to call it 'InstaVine'.

2. SYSTEM DESCRIPTION:

The system will consist of the following parts:

1. Camera for capturing the video

2. Video Processing DSP board

- The DIP switches on the board will be used for choosing the filter to be applied.
- The DSP chip will perform the necessary processing to apply the filter to each frame of the video at appropriate FPS

3. Monitor for displaying the real-time video with the filter applied. The advantage of this project over the already existing apps is that the effect of the filter can be seen as the video is being captured. In the existing apps, the filters are applied on captured and stored images or videos. A possible application of this project can be to integrate these features into digital cameras. This will allow the user to instantly capture videos with filters already applied to them.

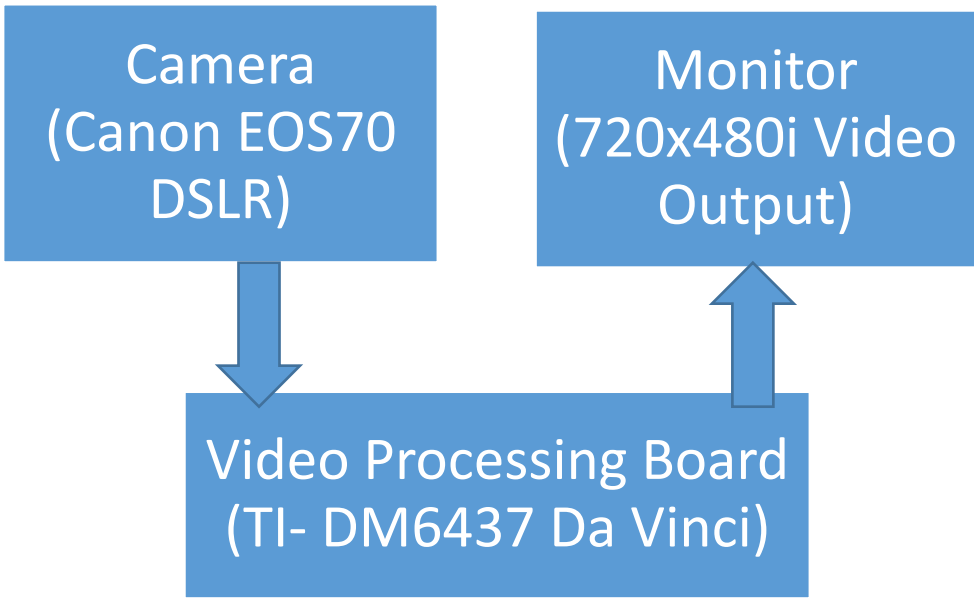


Fig [1]: Flow of the System.

2.1: CAMERA:

The Canon® EOS70 DSLR was used to capture real-time videos. The main advantages of using a DSLR were,

- High Definition recording as opposed to Standard definition outputs of the Lab camera
- Videos captures were less noisy and hence applying denoising was skipped
- Better ISO which helped in adjusting the lab lighting conditions during the demo.

The video feed from the camera was passed on to the TI DM6437 Da-Vinci board using a composite cable and further code on the board also ran on [NTSC] [Composite] mode.

2.2: VIDEO PROCESSING BOARD:

Fig [2] : DM6437 Da-Vinci Board

The salient features due to which this Video Processing Board was selected over others are as follows,

- DSP Frequency 600 MHz.
- Internal Memory (Fast) 256 Kbytes
- External Memory (Slow) 128 Mbytes DDR2 DRAM
- Input video resolution Interleaved 720 x 480. Output video resolution Interleaved 720 x 480.

- Video input interface RCA / S-video
- Video output interface RCA / S-video to external display which was not possible in 6416.

However, the caveat being, the main drawback of 6437 is smaller internal memory (256kB) which means we need to,

- Cache Input video frames efficiently. Block by Block processing wherever possible.
- Optimize the video processing code.

Colour space conversion from YCbCr to RGB and vice-versa was one of the most important aspects in this project. The data that we received from the camera was in YCbCr format and that had to be converted to RGB so as to get full colour information needed for most of our filters. And the RGB pixels needed to be converted to packed YCbCr again so as to render them on the external display.

Input YCbCr format: Input Buffer of the board represents one input frame which consists of two interleaved frames. Each 32 bit data has the information for two adjacent pixels in a row. Thus, the buffer holds 720/2 integer data points for each row of 2D image and there exist 480 rows (for a 720x480 frame). This data is represented by 4:2:2 sub-sampling scheme.

Format: yCbCr422 (y1 | Cr | y0 | Cb)

Each of y1, Cr, y0, Cb has 8 bits .For each pixel in the frame, it has y, Cb, Cr components.

Code snapshot for YCbCr -> RGB conversion:

```
for (i=0; i<Pixels; i++) {

	y0 = ((buffer_proc1[i] & 0xFF000000)>>24);//buffer_proc1 has input data
	y1= (buffer_proc1[i] & 0x0000FF00)>>8;
	cr = (buffer_proc1[i] & 0x00FF0000)>>16;
	cb = (buffer_proc1[i] & 0x000000FF);
	buf = buffer_proc1[i];

	r_temp1 = (10000 * (y0) + 13711 * (cr - 128));
	r_temp2 = (10000 * (y1) + 13711 * (cr - 128));
	g_temp1 = (10000 * (y0) - 3369 * (cb - 128) - 6892 * (cr - 128));
	g_temp2 = (10000 * (y1) - 3369 * (cb - 128) - 6892 * (cr - 128));
	b_temp1 = (10000 * (y0) + 17324 * (cb - 128));
	b_temp2 = (10000 * (y1) + 17324 * (cb - 128));
}
```

Code snapshot for RGB -> YCbCr packed conversion:

```
for (i=0;i<Pixels;i++) {

        buf = buffer_proc1[i];
        r_temp1 = r[2*i+0];
        g_temp1 = g[2*i+0];
        b_temp1 = b[2*i+0];
        y1 = ((2990*r[2*i+0]) + (5870*g[2*i+0]) + (1140*b[2*i+0]));
        cb1 = ((-1689*r[2*i+0]) - (3317*g[2*i+0]) + (5006*b[2*i+0]));
        cr1 = (4998*r[2*i+0]) - (4185*g[2*i+0]) - (812*b[2*i+0]);
        y2 = ((2990*r[2*i+1]) + (5870*g[2*i+1]) + (1140*b[2*i+1]));

        y1 = (int)((y1/10000)*219) << 7;
        cb1 = (int)(((cb1/10000)*224)<< 7)+128;
        cr1 = (int)(((cr1/10000)*224)<< 7)+128;
        y2 = (int)(((y2/10000)*219)<< 7);

        if(y1<0) y1 = 1;
        else if(y1>255) y1 = 255;
        if(y2<0) y2 = 1;
        else if(y2>255) y2 = 255;
        if(cb1<0) cb1 = 0;
        else if(cb1>255) cb1 = 255;
        if(cr1<0) cr1 = 0;
        else if(cr1>255) cr1 = 255;

      buffer_proc_inter[i] = ((y1 & 0x000000FF)<<24)|((cr1 &
      0x000000FF)<<16)|((y2 & 0x000000FF)<<8)|(cb1 & 0x000000FF);

}
```

3. IMPLEMENTED FILTERS:

3.1: CARTOON EFFECT:

3.1.1. ALGORITHM: Bilateral Filtering combined with edge map.

3.1.2. DESCRIPTION: The bilateral filter was chosen to implement the cartoon effect on our project. It is a technique to smooth images while preserving edges[1]. Following are the main aspects why bilateral filter is the filter of choice for smooth edge preserving effects.

- Its formulation is simple: each pixel is replaced by an average of its neighbours. This aspect is important because it makes it easy to acquire intuition about its behaviour, to adapt it to application-specific requirements, and to implement it.
- It depends only on two parameters that indicate the size and contrast of the features to preserve.
- It can be used in a non-iterative manner. This makes the parameters easy to set since their effect is not cumulative over several iterations.

Convolution by a positive kernel is the basic operation in linear image filtering. It amounts to estimate at each position a local average of intensities and corresponds to low-pass filtering. The Gaussian blur of an image 'I' can be represented as,

$$Gaussian\ Blurred[I] = \sum_{i=0}^{col-1} \sum_{j=0}^{row-1} G_\sigma\left(|i-j|\right)\ I[i][j]$$

$$where, G_\sigma\left(x\right) = \frac{1}{2\pi\sigma^2}\exp\left(\frac{-x^2}{2\sigma^2}\right)$$

So, Gaussian filtering is a weighted average of the intensity of the adjacent positions with a weight decreasing with the spatial distance to the center position. This distance is defined by $G_\sigma(|i-j|)$ where σ is a parameter defining the extension of the neighbourhood. As a result, image edges are blurred.

Similarly to the Gaussian convolution, the bilateral filter is also defined as a weighted average of pixels. The difference is that the bilateral filter takes into account the variation of intensities to preserve edges. The rationale of bilateral filtering is that two

pixels are close to each other not only if they occupy nearby spatial locations but also if they have some similarity in the photometric range.

$$Bilateral[I] = \frac{1}{W_p} \sum_{i=0}^{col-1} \sum_{j=0}^{row-1} G_{\sigma s}(||i-j||)\, G_{\sigma r}(Ii - Ij)\, I[i][j]$$

$$where, Wp\ is\ the\ normalization\ factor;$$

$$Wp = \sum_{i=0}^{col-1} \sum_{j=0}^{row-1} G_{\sigma s}(||i-j||)\, G_{\sigma r}(Ii - Ij)$$

Parameters $\sigma_s$and σ_r will measure the amount of filtering for the image I. Equation (2) is a normalized weighted average where Gσs is a spatial Gaussian that decreases the influence of distant pixels, Gσr a range Gaussian that decreases the influence of pixels q with an intensity value different from Ip. Note that the term range qualifies quantities related to pixel values, by opposition to space which refers to pixel location.

3.1.3. Effect of Range and Spread Parameters: The bilateral filter is controlled by two parameters: the spread parameter σs and the range parameter σr

- As the range parameter σr increases, the bilateral filter becomes closer to Gaussian blur because the range Gaussian is flatter i.e., almost a constant over the intensity interval covered by the image.
- Increasing the spatial parameter σs smooths larger features.
- An important characteristic of bilateral filtering is that the weights are multiplied, which implies that as soon as one of the weight is close to 0, no smoothing occurs.
- As an example, a large spatial Gaussian coupled with narrow range Gaussian achieves a limited smoothing although the filter has large spatial extent.
- The range weight enforces a strict preservation of the contours.

3.1.4. APPLICATIONS:

Following are some of the challenging applications in which the bilateral filter has been used:

- **Denoising**: This is of course the primary goal of bilateral filter, and it has been used in several applications such as medical images, movie restoration, etc. For instance, Adobe Photoshop® provides the bilateral filter under the name ”surface blur”. It uses a square box function as spatial weight and a tent function as range weight. Unlike Gaussian blur that smoothes images without respecting their visual structure, the bilateral filter preserves the object contours and produces sharp results. The adapting the range parameter σr to the local noise level yields more satisfying results. In practice, they advise a linear dependence: σr = 1.95 σn where σn is the local noise level. Although, bilateral filtering preserves edges, the preservation is not perfect and some edges are sharpened during process, incurring an undesirable clustering of edges and contours.

- **Texture and Illumination Separation, Tone Mapping, Retinex, and Tone Management**: Based on a large-scale / small-scale decomposition of images, these applications edit texture and manipulate the tonal distribution of an image to match the capacities of a given display or achieve photographic stylization. Use of bilateral filtering can be extended to isolate small-scale signal variations including texture but also small details of an image. Tone mapping, whose goal is to compress the intensity values of an high-dynamic range image to visualize it on a low-dynamic range display. Naive solutions such as uniform scaling or gamma correction yields unsatisfactory results since scene details are lost because of intensity compression. One possible solution is to isolate the details before compressing the intensity. Apply the bilateral filter on the log-intensities of the HDR image, scale down uniformly the result, and add back the filter residual, thereby ensuring that the small-scale details have not been compressed during the process.

- **Three-dimensional Fairing**: This is the counterpart of image denoising for three-dimensional meshes and point clouds. Noise is removed from these data sets.

- **Depth Map from Luminance**: Bilateral filtering can also be used to process the luminance channel of an image and obtain a pseudo-depth map that is sufficient for altering the material appearance of the observed object. The originality of this use of the bilateral filter is that the smoothing power of the bilateral filter determines the geometric characteristics of an object. For instance, a smaller intensity tolerance σr results in a depth map that looks like “engrave” with the object texture because the intensity patterns are well preserved and directly transferred as depth variations.

- **Video Stylization and Cartoon Effect**: Iterate the bilateral filter in order to simplify video content and achieve a cartoon look. Bilateral filtering is an effective preprocessing for edge detection: filtered images trigger fewer spurious edges. To modulate the smoothing strength of the bilateral filter, the degree of edge preservation can be controlled. The range weight Gσr is replaced by (1 – m). Gσr + m, where m is a function varying between 0 and 1 to control edge preservation, and u defines the local importance of the image. The same theory is used to achieve the cartoon effect in InstaVine. The only difference being, in order to accentuate the edges, we use the adaptive canny edge detection algorithm in conjugation with the Bilateral filtering on the color channels.

3.1.5. IMPLEMENTATION:

The following are the most commonly used implementations of the Bilateral Filter,

- **Brute Force Implementation**:

  For each pixel p in the Image frame 'I';

  1. Initialization: Ip = 0, Wp = 0;
  2. For each pixel q in the Image 'I'.
     (a). $w = G\sigma_s\left(|| p - q ||\right) G\sigma_r\left(\left|| I_p - I_q \right|| \right)$
     (b). $I_p \mathrel{+}= wI_q$
     (c). $W_p \mathrel{+}= w$
  3. Normalization: $I_p = \frac{I_p}{W_p}$

- **Separable Kernels**: We improved the time complexity of the brute force method by restricting the inner loop to the neighbourhood of the pixel p. Typically, one considers only the pixels q such that ||(p-q)||< ($2\sigma_s$). The rationale is that the contributions of pixels farther away than $2\sigma_s$ is negligible because of the spatial Gaussian. This leads to a complexity on the order of O (Total Pixels * $\sigma_s * \sigma_s$).This implementation is efficient for small spatial kernels, that is, small values of σ_s but becomes quickly prohibitive for large kernels because of the quadratic dependence in σ_s.[2]. This approach yields significantly faster running times but the performance still degrades linearly with the kernel size.

3.2. COMIC BOOK EFFECT:

3.2.1. ALGORITHM: Dithering, Error Diffusion, Sharpening filter with Brightness retention.

3.2.2. DESCRIPTION: To generate a comic book like printing effect, it was necessary to half-tone or decreases the color gamut of the image pixels intensities so as to give the 'printed' effect. Dithering and error diffusion are the main methods to implement color half-toning. Moreover, in order to give the comic book effect, the edges needed to be black in color and needed to be accentuated. For this purpose, image sharpening was employed. Halftoning is a thus, a method for creating the illusion of continuous tone output with a binary device[3].

3.2.3. DITHERING: The concept of dithering is related to printing of various colour /greyscale images. Sometimes, the dynamic range and colour variations of the images are so high that it exceeds the printer's ability to print it faithfully. Hence, the amount of colour/grey level information needs to be reduced in order to print it. Dithering is one such method to reduce the colour/grey level information within a particular image. It also refers to reproducing the grey scale by increasing/decreasing the spatial resolution of the ink dots that are printed on the paper[4]. And this generated the '*COMIC BOOK*' effect. In dithering matrix case, an index matrix is used to render pixels of an input image into different pixel intensity level depending upon the spatial distance of that pixel from the pixel of interest. The index matrix can be 2x2 or higher orders. Thus the rendered pixel intensity depends upon the values of the index matrix which indicate how likely the dot will be turned on/off. The value 0 indicated that the likelihood of the pixel being turned on is more than as compared to 3. This is done on all the three RGB channels in case of coloured dithering. The following a 2x2 index matrix.

$$I_2(i,j) = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}$$

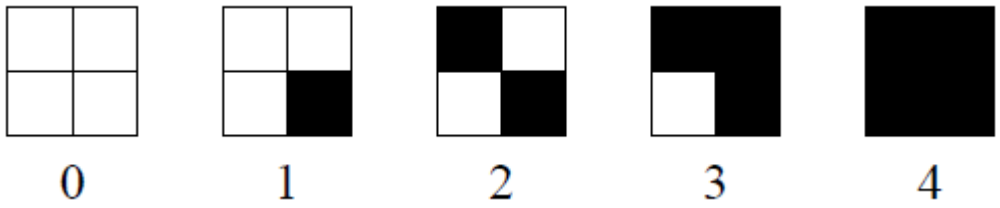


Fig [3]: Order of Pixel turning on/off.

Higher order Dithering matrices can be obtained from the basic 2x2 matrix by the formula as follows;

$$I_{2n}(i,j) = \begin{bmatrix} 4 * I_n(x,y) + 1 & 4 * I_n(x,y) + 2 \\ 4 * I_n(x,y) + 3 & 4 * I_n(x,y) + 0 \end{bmatrix}$$

Thus the higher order index matrices are;

$$I_4(x,y) = \begin{bmatrix} 5 & 9 & 6 & 10 \\ 13 & 1 & 14 & 2 \\ 7 & 11 & 4 & 8 \\ 15 & 3 & 12 & 0 \end{bmatrix}$$

The corresponding Thresholding elements are given by the formula;

$$T(x,y) = 255 * \left(\frac{I(x,y) + 0.5}{N * N} \right)$$

Thus a space varying threshold is given as, $T(i,j)$ for all the three channels where,

$$b(i,j) = \begin{cases} 255\ ; & I(i,j) > T(i,j) \\ 0\ ; & otherwise \end{cases}$$

Code Snapshot: The following is the code snapshot of dithering by 4x4 Thresholding;

```
for(unsigned int i=0;i<height;i=i+4){
      for(unsigned int j=0;j<width; j=j+4){
            for(unsigned int h=0;h<4;h++){
                  for(unsigned int k=0;k<4;k++){

                        if((Imagedata[i+h][j+k] > T44[h][k])
                              Dither44Image[i+h][j+k]  = 255;
                        else
                              Dither44Image[i+h][j+k] = 0;
                  }
            }
      }
 }
```

Disadvantages of Dithering: The disadvantage of dithering is that it introduces artificial repetitive patterns caused by the Thresholding matrix. These are the low frequency effects observed in the dithered image.

3.2.4. ERROR DIFFUSION: Here, each pixel is quantized using a neighbourhood operation, rather than a simple point-wise operation. This can be done by moving along the image in raster scanning fashion or by serpentine scanning order. These low frequency effects can be removed by using the 'blue noise' effect by artificially creating noise and use that to our advantage by feedback looping to diffuse that quantization error obtained after Thresholding to right neighbouring pixels.

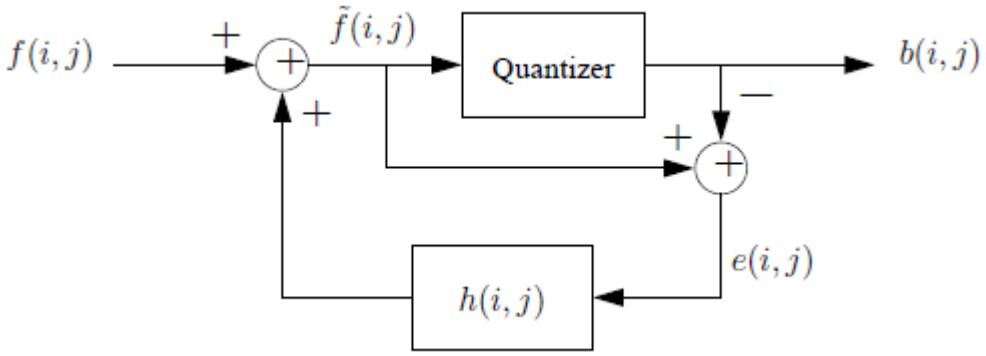


Fig [4]: Filter View of the Error Diffusion process.

Here,

$$b(i,j) = \begin{cases} 255\,; & \tilde{f}(i,j) > T \\ 0\;\; ; & otherwise \end{cases}$$

$$e(i,j) = \tilde{f}(i,j) - b(i,j)$$

And,

$$\tilde{f}(i,j) = f(i,j) + \sum_{k=0}^{row-1} \sum_{l=0}^{col-1} h(k,l)\, e(i-k, j-l)$$

In this project, the errors are diffused forward to next pixel to be processed.

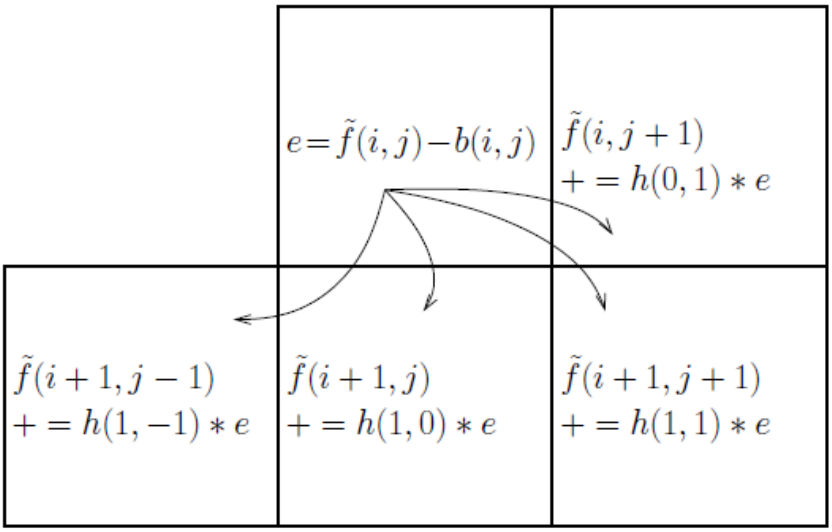


Fig [5]: Forward error diffusion mechanism.

In the project, the Floyd-Steinberg forward error diffusion[5] mechanism with serpentine scanning was used so as to prevent the error to get diffused to just one side of the image.

		7/16
3/16	5/16	1/16

Fig [6]: Floyd-Stenberg error diffusion matrix.

3.2.5. SHARPENING FILTER: To sharpen the image is very similar to finding edges, add the original image, and the image after the edge detection to each other, and the result will be a new image where the edges are enhanced, making it look sharper. Adding those two images is done by taking the edge detection filter and incrementing the center value of it with 1. Now the sum of the filter elements is 1 and the result will be an image with the same brightness as the original, but sharper.

Code Snapshot for a sharpening convolution matrix:

```
int Sharpening_matrix [Width][Height] = {   -1, -1, -1,
                                            -1,  9, -1,
                                            -1, -1, -1 };
Unsigned int factor = 3;
Unsigned int bias = 1;
```

3.3. COLOR TEMPERATURE EFFECT:

3.3.1. ALGORITHM: Colour space correlation with temperature[6].

3.3.2. DESCRIPTION: The "color temperature" of an image is simply the red or blue cast to an image which can indirectly be related to the color of a star based on its blackbody temperature.

- The color curves shown below represent those temperatures, with the white-point being held at approximately 6600k.
- Color temperature effects are often used during video or photo production in order to set a particular tone for the artwork, such as the blue optical filters used during production of the twilight movie series.
- Color temperature modifications are achieved by first calculating the RGB values corresponding to the desired temperature. These values are then averaged with each pixel in the image and converted to HSL color space.
- The hue and saturation values are used, while luminance from the original image replaces the new one. This new image can then be converted back to RGB color space, and then YCbCr for output.

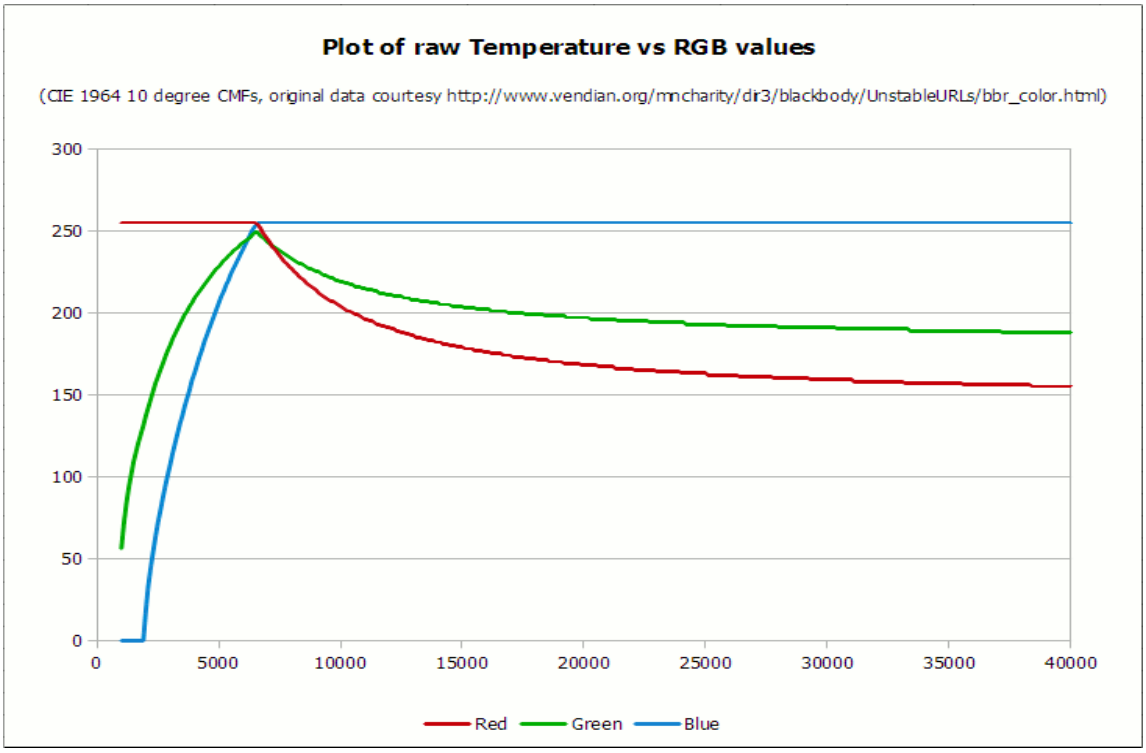


Fig [7]: Temperature to RGB mapping.

Fig [8]: Temperature display from 1500K to 15000K.

3.3.3. PSEUDO-CODE:

```
Set Temperature = Temperature \ 100

    Calculate Red:

    If Temperature <= 66 Then
        Red = 255
    Else
        Red = Temperature - 60
        Red = 329.698727446 * (Red ^ -0.1332047592)
        If Red < 0 Then Red = 0
        If Red > 255 Then Red = 255
    End If

    Calculate Green:

    If Temperature <= 66 Then
        Green = Temperature
        Green = 99.4708025861 * ln(Green) - 161.1195681661
        If Green < 0 Then Green = 0
        If Green > 255 Then Green = 255
    Else
        Green = Temperature - 60
        Green = 288.1221695283 * (Green ^ -0.0755148492)
        If Green < 0 Then Green = 0
        If Green > 255 Then Green = 255
    End If

    Calculate Blue:

    If Temperature >= 66 Then
        Blue = 255
    Else
```

```
        If Temperature <= 19 Then
            Blue = 0
        Else
            Blue = Temperature - 10
            Blue = 138.5177312231 * ln(Blue) - 305.0447927307
            If Blue < 0 Then Blue = 0
            If Blue > 255 Then Blue = 255
        End If

    End If
```

3.4. WHITE BALANCE CORRECTION:

3.4.1. ALGORITHM: Histogram mapping and component stretching assuming that the background will be non-biased with respect to one colour[7].
3.4.2. DESCRIPTION: Many forms of white balance correction exist, however very few are known to work without extensive training for specific conditions. Three main methods are known to work well in non-constrained conditions,

- **White-patch method:** The white-patch method assumes that the brightest pixels in an image can be remapped to pure white where r=g=b, with the rest of the pixels being remapped proportionally. This method was not viable due to the input video feed containing an overlay which would skew colour results.

- **Grey-world method**: The grey-world method assumes that the image has a good distribution of colors, and that the average reflected color should be the color of the light source, which can be assumed to be white. This method was initially tested in RGB color space and provided poor results, therefore this method was discarded.

- **Component stretching**: White balance correction in this project uses this method which will be discussed in detail as follows,

Component stretching method stretches each colour histogram to its full range in order to create a colour balanced image. The bottom 1.4% and top 0.7% of pixels are ignored to help account for black/white pixels in the camera video overlay. Component stretching is sensitive only to large single-colour regions in the image, such as a red wall/background; however it was implemented with the assumption that the background would be non-colour-biased and somewhat dynamic. Additional study has

suggested that this method could provide the best results when operating in YUV colour space and could be implemented in the future for comparison.

3.5. SEPIA TONE

3.5.1 ALGORITHM: Color channel cross-mixing

3.5.2 DESCRIPTION: A sepia tone can be generated similarly to a grayscale image, however using only a simple ratio of RGB values does not deliver an image with appealing contrast or dynamic range. In order to provide a visually pleasing image, the color channels are cross-mixed (cross-multiplied) and renormalized. These cross-mixed channels are then used to grab values from a gamma look-up table in order to produce sepia-toned images with a high dynamic range using a gamma of 1.6.

3.6. DISSOLVE

3.6.1. ALGORITHM: Localized pixel re-distribution[8]

3.6.2. DESCRIPTION: The dissolving effect is a simple but elegant filter. The function is given a maximum range for dispersion of pixels, and within that range will swap pixels at random distances in each direction for every pixel. This is applied with a looping set of ranges in order to display how it could be used for a fade-in or fade-out effect.

3.7 PENCIL SKETCH EFFECT

3.7.1. ALGORITHM: Adaptive Canny Edge Detection Algorithm[9].

3.7.2. DESCRIPTION: The following steps characterize the implementation of the pencil sketch using adaptive Canny.

- Extract the luminance information from the frame of the video (image). This is done by masking the Cb and Cr components of the interlaced 4:2:2 YCbCr packed data of the image.
- Smooth the image and remove noise by Gaussian filtering the image
- Using Sobel filter, compute the horizontal derivative Gx and vertical directional derivative Gy of the image.
- Using Gx and Gy, calculate gradient magnitudes: $\nabla f = \sqrt{Gx^2 + Gy^2}$
- Calculate the gradient direction.

- Perform Non-Maxima Suppression (NMS) to the gradient magnitudes image. This ensures that there is only pixel in the thickness of the edge. This also ensures that there are no broken edges.
- A pixel is deemed to be a possible edge if the gradient magnitude is greater than the gradient magnitude of the other two points along the direction of the current 3x3 neighborhoods and then set its sign to 1. Otherwise the current pixel is a non-edge pixel and it is set to 0. We thus get a map of possible edge pixels[10].
- Calculate gradient histogram difference diagram analysis method
- To find the suitable maximum value in Intensity histogram, we use the grads threshold method. We first find the differential:
  $diff(i) = diff\ (i) - diff\ (i-1)$, where diff(i) is the number of differential pixels in the gradient histogram difference diagram.
- Gradient Histogram difference diagram is then plotted using the following : $(diff(i-1) + diff(i) > 0)\ OR\ (\ diff(i) + diff(i+1) > 0\ )$
- The image is then classified into one of the three categories based on the amount of edge information in it
- Little edge information, rich edge information or numerous edge information
- The classification is done based on the number of pixels whose gradient magnitude is greater than zero.
- Little edge information – when the amount of edge information is less than five times the perimeter of the image.
- Rich edge information – when the amount of edge information is between five times and twenty times the perimeter of the image
- Numerous edge information – when the amount of edge information in the image is more than twenty times the perimeter of the image.

- For little edge information :

  $TH_h = \min\big(\ i :\ \ i \in (2, THmax\ )\ AND\ \ f)\big)\ \ ;\ \mathrm{TH_l} = \mathrm{TH_h}/2$

  Where TH_h= high threshold, TH_l = low threshold, $THmax$ is the maximum possible threshold, which is the highest value of pixel I within the frame for which f(i) = 1.

- For rich edge information:

$$TH_h = \max(\ i : i \in (3, THmax) AND\ f\ AND\ Dif^{th} >\ Dif_{5C}\ )$$

$$TH_h = \max(\ i : i \in (3, THmax) AND\ f\ AND\ Dif^{th} >\ Dif_{Total}\ /2\ )$$

Where, Dif^{th}= sum of pixels whose gradient is larger than TH_hor TH_h, Dif_{5C} = 5 x perimeter of the image, Dif_{Total}= number of pixels which are possible edges of the image

- For numerous edge information:

$$\mathrm{TH_h} = \max(\,\mathrm{i} : \mathrm{i} \in (3, \mathrm{THmax}) \mathrm{AND\ f\ AND\ Dif^{th}} > \mathrm{Dif_{Total}}\ /12\ )$$
$$\mathrm{TH_l} = \max(\,\mathrm{i} : \mathrm{i} \in (3, \mathrm{THmax}) \mathrm{AND\ f\ AND\ Dif^{th}} > \mathrm{Dif_{Total}}\ /4\ )$$

- After calculation of high threshold and low threshold, the edges are found by classifying a pixel as edge if the pixel value is greater than high threshold. Also, if any of the neighboring pixels in the direction of the gradient direction map is greater than the low threshold, then that pixel is classified as an edge too.

3.8 VIGNETTE

3.8.1. ALGORITHM: Skin Colour Segmentation with faded mask overlay

3.8.2. DESCRIPTION: The following process was carried out for Vignette using skin colour segmentation,

For Skin colour Segmentation;

- Convert the YCbCr data to RGB.
- Use the RGB data to calculate the Hue component of HSV color space using the following

```
for (int i=0;i<height;i++){
     for (int j=0;j<width;j++) {

          int min;    //Min. value of RGB
          int max;    //Max. value of RGB
          int delMax; //Delta RGB value

          r = (img(j,i)>>16) & 0xFF;
          g = (img(j,i)>>8) & 0xFF;
          b = (img(j,i)) & 0xFF;

          if (r > g) { min = g; max = r; }
          else { min = r; max = g; }
          if (b > max) max = b;
          if (b < min) min = b;
```

```
            delMax = max - min;

            float Hue = 0;
            float Sat = 0;
            float Val = max;

            if ( delMax == 0 ) { Hue = 0; Sat = 0; }
            else {
                  Sat = delMax/255;
                  if ( r == max )
                        Hue = ((g - b)/(float)delMax)*60;
                  else if ( g == max )
                        Hue = (2 +  (b - r)/(float)delMax)*60;
                  else if ( b == max )
                        Hue = (4 +  (r - g)/(float)delMax)*60;
      }
}
```

Iterate through the image and whichever pixel falls within the thresholds for skin color tones, mark that as 1 (or 255) and the rest as 0 (black).

The thresholds[11] used are:

- The first thresholding is done on Hue: where Hue < 25 AND Hue > 230
- The next thresholding is done on the Cb and Cr color channels where
    - Cr <= 1.5862 Cb + 20
    - Cr >= 0.3448 Cb + 76.2069
    - Cr >= -4.5652 Cb + 234.56
    - Cr <= -1.15 Cb + 301.75
    - Cr <= -2.2857 Cb + 432.85

- All of these conditions need to be met at the same time for a pixel to be classified as skin color.
- After this, the thresholds are applied on the RGB values. For the RGB color space, the thresholds are as follows:
- Skin color at uniform daylight illumination has the following thresholds:

$$(R > 95\ )AND\ (G > 40)AND\ (B > 20)$$
$$AND\ (\max\{R,G,B\} - \min\{R,G,B\} > 15)$$
$$AND\ (|R - G| > 15\ AND\ (R > G)AND\ (R > B)\ )$$

- Or the skin color under lateral illumination has the following thresholds:

$$(R > 220\ )AND\ (G > 210)AND\ (B > 170)AND$$
$$(|R - G| \leq 15\ AND\ (R > B)AND\ (G > B)\ )$$

If the RGB values fall under at least one of these RGB buckets, they are classified as skin color.

For Faded Mask Overlay;

- Once the binary image with all the skin colored portions as white and the rest of the regions as white has been prepared, we can calculate the density of white pixels in each column.
- The area where the white pixels are more concentrated in many columns in the vicinity of each other, that region is assumed to be the face of the person in the frame.
- A bounding box is drawn around the frame and the center is found.
- This center point is used as the center for the fading of the mask that is to be overlaid on top of the image.
- The mask is generated as a separate image using

$$f(i,j) = radius * X^{grade}$$

Radius and grade are parameters that are predefined. (radius = 2 and grade = 2) with

$$f(i,j) = \frac{|L - Ln|}{L}$$

where

$$L = \sqrt{xc^2 + yc^2}$$
$$Ln = \sqrt{(xc - i)^2 + (yc - j)^2}$$

3.9 EMBOSS EFFECT

3.9.1 ALGORITHM: Bump Map of the frame with pixel intensity level shifting.

3.9.2. DESCRIPTION: The emboss filter gives a 3D shadow effect to the image, essentially by taking the bump-map of the frames along the diagonal.

This filter stamps and carves the active layer or selection, giving it relief with bumps and hollows. Bright areas are raised and dark ones are carved[12]. In our implementation we

have focused only on bump-maps along the diagonal 45deg direction of the image, the effect of the azimuth, elevation and depth changes are not implemented.

- Bump-maps store an intensity that is relative to the height of pixels from the viewpoint of the camera. The pixels seem to be moved by the required distance in the direction of the face normals.
- The "bump" consists only of a displacement, which takes place along the existing, and unchanged, normal-vector of the face.
- Bump height depends on pixel luminosity and the magnitude of the edge strength at that location.
- Pixel intensity offset of 127 is added to all the pixel values output from the embossing convolution matrix to colour the embossed image grey. This gives a better 3D visual embossing effect.

3.9.3. Code Snapshot for a sharpening convolution matrix:

```
int Emboss_matrix [Width][Height] = {       -1, -1,  0,
                                            -1,  0, -1,
                                             0, -1, -1 };
Unsigned int factor = 1;
Unsigned int bias = 127;
```

And a more accentuated bump-map is created by

```
int Emboss_matrix [Width][Height] = { -1, -1, -1, -1,  0,
                                      -1, -1, -1,  0,  1,
                                      -1, -1,  0,  1,  1,
                                      -1,  0,  1,  1,  1,
                                       0,  1,  1,  1,  1};
Unsigned int factor = 1;
Unsigned int bias = 127;
```

4. RESULTS:

Following are the timing results from our demo;

Filter	Frames/Second
Edge Detection	0.17
White Balance	2.1
Sepia	3.5
Color temperature	2.9
Grayscale Vignette	0.15
Black Vignette	0.16
Emboss	1.6
Cartoon	1.2
Dissolve	1.55

Table[1] : Timing Results

Following are the screenshots from our demo

Fig [9] : White Balance Correction

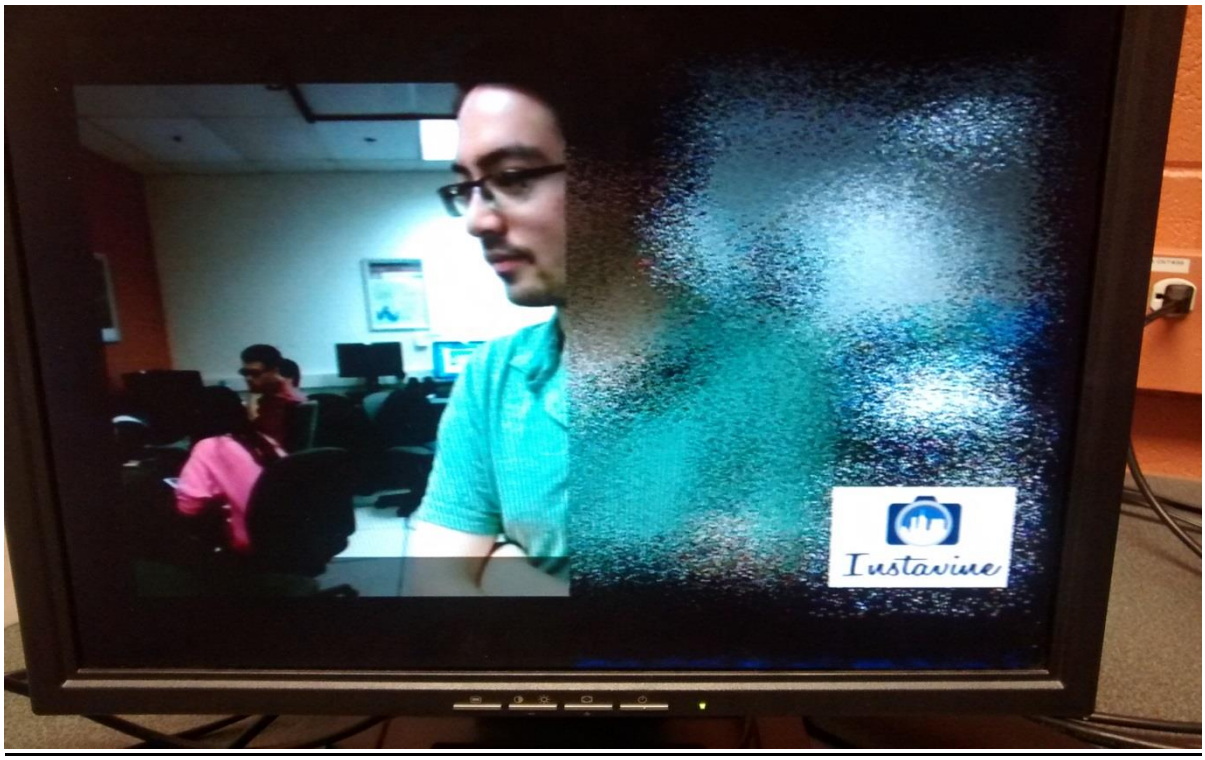


Fig [10]: Dissolve effect

Fig [11] : Pencil Sketch effect

Fig [12] : Vignette (Focus) effect

Fig [13]: Vignette (Greyscale) effect

Fig [14]: Emboss effect

Fig [15]: Comic Book effect

Fig [16]: Sepia Tone effect

Fig [17]: Colour Temperature effect

5. CODE OPTIMIZATIONS:

Some simple techniques were employed in order to greatly increase the processing speed of the real-time filters.

- The most significant impact was gained from converting floating-point calculations to integer operations, which can tremendously increase speed on any non-floating point processer.

- When floating point values were required, such as for exponential or logarithmic calculations, lookup tables were generated for each frame. This generally resulted in only a few hundred calculations per frame instead multiple calculations per pixel.

- Individual pixel values or windows were also pre-fetched from DDR and placed into variables held in the cache in order to speed up local-processing times and reduce the impact of using external memory.

- Loop unrolling wherever possible.

- Code composer studio's in built code optimization during build was also used.

6. CHALLENGES:

- Some variables would often become corrupted on the DSP board despite being managed correctly in the code; this was likely a stack issue when calling functions. The issue was resolved by simply using unique variable names, ensuring that the registers would update.
- The board would occasionally not update to the current build. This could sometime be resolved by power-cycling the board, however it would occasionally require a fresh project, cleaning the project rarely helped.
- Internal memory was very limited on the TI DM6437, therefore the external SDRAM was utilized to hold all large arrays of data. The pragma definition did not always force arrays into SDRAM, therefore the linker file was changed to define all .far variables in SDRAM.
- Even after optimizing this process, the main bottleneck was still this DDR R/W cycle.
- YCbCr to RGB conversion has many possible solutions published, most of which caused overflows, resulting in artifacts in the image. In order to avoid any overflows, both color spaces were restricted to values from 16 to 240, resulting in a clean image.
- The main challenge of the bilateral filter is its time complexity and computational cost. The TI DM6437 board with 600MHz clock speed is far too slow to handle real time video processing using Bilateral filter and it takes a few 10's of seconds to process on one frame. As the result the input buffer for the incoming video frames overflow and the processor eventually runs out of memory to hold-process and render video frames. Hence, even though the filter worked on our C platform, the issues faced with real time implementation on the board far exceeded our expectations and the filter could not be realized on DM 6437 correctly.
- For the edge detection using adaptive canny, the categorization parameters given in the paper referred were given as:

  Little edge information – when the amount of edge information is less than five times the perimeter of the image.

  Rich edge information – when the amount of edge information is between five times and twenty times the perimeter of the image

  Numerous edge information – when the amount of edge information in the image is more than twenty times the perimeter of the image.

However, these values did not work very well with our setup. As a result, all the frames being processed were being categorized as little edge information category. Hence, we increased the categorization parameters by ten times.

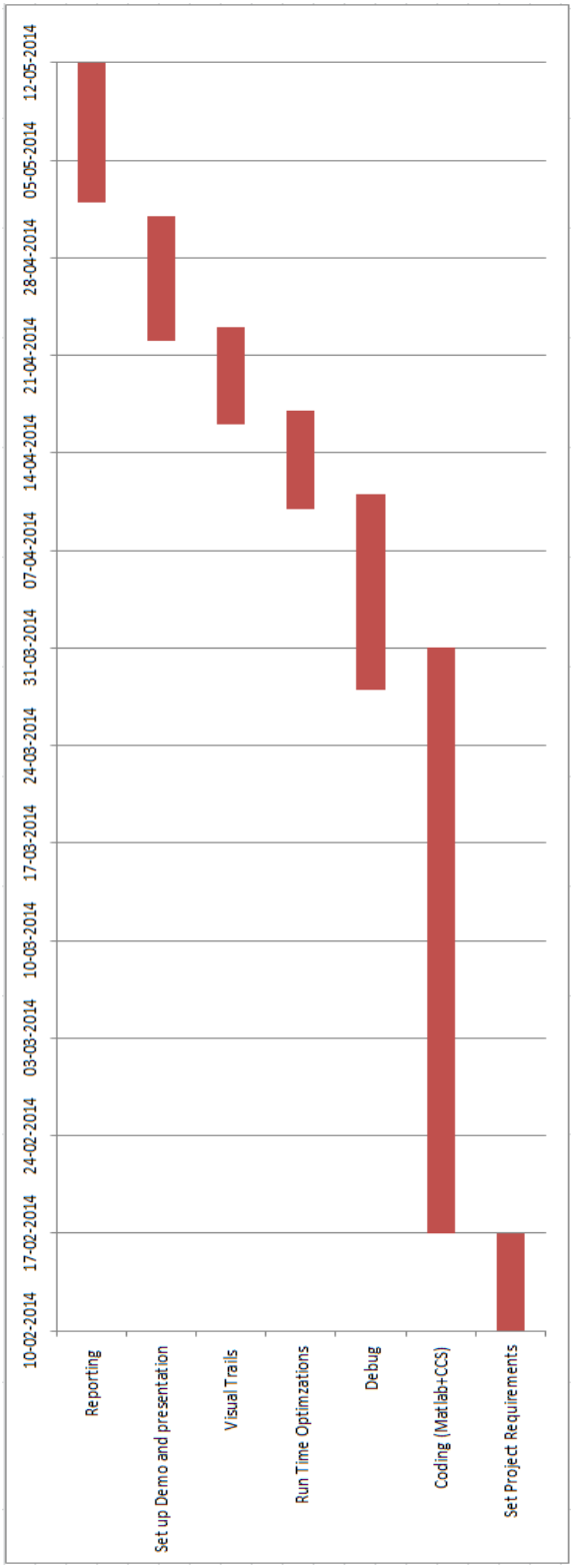
- For skin color segmentation – the drawback is that the algorithm can only detect one person per frame. Since, it detects the face based on density, if there are two people in the frame, the one with more visible skin area would be detected. Since, most people have similar sized faces, the algorithm keeps shifting the focus from one person to the other. More robust algorithm is required for this purpose. However, for our application, getting into complex face detection algorithms like motion vectors or Viola-Jones method was not computationally feasible.

7. CHECK LIST:

Proposed	Delivered
Pencil Sketch	Pencil Sketch using Adaptive Canny
Oil Painting Effect	Cartoonize
Comic Book Effect	Comic book (Sharpening+Edge with Brightness retention)
White Balance Correction	White Balance Correction using component analysis
Vignette	Vignette with skin segmentation - GreyScale - Focus
Color Temperature	Warm to Cool Color temperature effect
Sepia	Sepia
	Metallic Emboss effect
	Dissolve Filter

Table[2] : Checklist

8. SCHEDULE AND WORK DISTRIBUTION:



Table[3]: Gantt Chart of the schedule

Effect	MATLAB Code - Written By
Pencil Sketch	Shruti
Cartoon Effect	Manan
Comic Book Effect	Manan
White Balance Correction	Nicholas
Vignette	Manan
Color Temperature	Nicholas
Sepia	Nicholas
	C-Code for CCS Written by
Pencil Sketch	Shruti
Emboss effect	Manan
Comic Book Effect	Manan
White Balance Correction	Nicholas
Vignette - Grey Scale	Shruti
Vignette - Focus	Shruti
Color Temperature	Nicholas
Sepia	Nicholas
Dissolve	Nicholas
Miscelleneous	
Report	Material In Collaboration ; Final Compilation by Manan(Final) and Shruti(Proposal)
Presentations	In Collaboration
Instavine logo	Manan
Optimizations	Nicholas

Table[4] : Responsibilities of group members

9. FUTURE SCOPE:

- Include Motion Vector effects and Video compression in our project.
- User defined filters with options to control the amount of effect to be added by some sort of UI.
- Developing an Android app on similar lines.
- Sell the product to YouTube/ DailyMotion ☺
- Facility to plug in a memory device and be able to simultaneously store the output in the memory device.
- Load or stream a video from online source, apply the filter and again stream it back.

REFERENCES:

[1] C. Tomasi and R. Manduchi, "Bilateral Filtering for Gray and Color Images", *Proceedings of the 1998 IEEE International Conference on Computer Vision*, Bombay, India.

[2] K.N. Chaudhury, D. Sage, and M. Unser, "Fast O(1) bilateral filtering using trigonometric range kernels," *IEEE Trans. Image Processing, vol. 20, no. 11*, 2011.

[3] A.U.; Allebach J.P. Baqai, F.A.; Je-Ho Lee; Agar. Digital color halftoning. In Signal Processing Magazine,IEEE,vol. 22, pages 87,96, Jan 2005.

[4] T. Mitsa and K.J. Parker, "Digital halftoning technique using a blue-noise-mask," J. Opt. Soc. Am. A, vol. 9, pp. 1920-1929, 1992.

[5] R.W. Floyd and L. Steinberg, "An adaptive algorithm for spatial grayscale", Proc. SID, vol. 17/2, pp. 75-77, 1976

[6] <online> http://www.tannerhelland.com/4435/convert-temperature-rgb-algorithm-code/ dated 10-May-2014

[7] Edmund Y. Lam, *Member*, *IEEE,* Combining Gray World and Retinex Theory for Automatic White Balance in Digital Photography, *Consumer Electronics, 2005. (ISCE 2005). Proceedings of the Ninth International Symposium 134 - 139,* 14-16 June 2005.

[8] <online> http://www.tannerhelland.com/3601/realtime-diffuse-spread-image-filter-vb6/ dated 10-May-2014

[9] Luo Tao; Xi feng Zheng; Ding Tie-fu. Self-adaptive threshold canny operator in color image edge detection. In *2nd International Congress on Image and Signal Processing, CISP '09*, pages 1,4, 17-19 October 2009.

[10] Y.; Cetin A.E. Akarun, L.; Yardunci. Canny edge detection codec using vlib on davinci series dsp.In *International Conference on Computer Science and Service System (CSSS),* pages 221,224, 11-13 August 2012.

[11] Nusirwan Anwar bin Abdul Rahman, Kit Chong Wei and John See. RGB-H-CbCr Skin Colour Model for Human Face Detection.

[12] Schlag, "Fast Embossing Effects on Raster Image Data", Graphics Gems IV, AP Professional, 1994, gem VIII.1, pp. 433-437